Vol.2018-DPS-176 No.11
Vol.2018-EIP-81 No.11
2018/9/13

Rust から Verilog へのトランスパイラの実装
Implementation of Rust to Verilog Transpiler

高野 恵輔† 小田 哲也‡ 小畑 正貴‡
Keiuske Takano Tetsuya Oda Masaki Kohata

1. はじめに

近年、ストリーミング処理や、ディープラーニングなどの計算量の大きい処理が多用されている。これらの処理は計算量に比例して処理時間も大きくなるため高速化を可能にする演算器が求められている。このような状況で、処理の高速化や電力消費量の削減が可能であるとして FPGA(Field Programmable Gate Array)が注目されている。しかしながら、FPGA を開発する際に使用される Verilog 等の HDL(Hardware Description Language)では開発時間が長期化するだけでなくメモリ安全を保証することが困難である。そこで本研究では、メモリ安全性を有する言語である Rust によってアルゴリズムを記述し、HDL のうち Verilog へと変換するトランスパイラを実装した。

2. トランスパイラ

トランスパイラとは、あるプログラミング言語で記述されたソースコードを入力として受け取り、別のプログラミング言語の同等なコードを目的コードとして出力するツールである。提案システムは Rust から Verilog に変換するトランスパイラである。

Rust[1]とは Mozilla が開発を支援するオープンソースのシステムプログラミング言語である。コンパイラが、リソースの静的検証を事前に行い、異なる型の変数間での代入、また不正なメモリ領域へのアクセスなどを防ぐことにより型間やメモリ安全なバイナリを生成することで実現している。

3. 提案システム[2]

本提案システムは Rust を用いたトランスパイラである。Verilog の構文に対応させた構造体を定義することで Rust 上に Verilog のための AST(Abstruct Syntax Tree)実装を行う。

AST とは、抽象構文木のことで対象とした言語にとって意味のない情報を取り除き、意味に関係のある情報のみを抽出した木構造のデータ構造である。コンパイラやインタプリタ内の中間処理として用いられる。提案システムでは Verilog の AST を Rust 上で構築することで機能を実現する。図 1 に AST 構築の手順を示す。Verilog の module と対応させた VModule という構造体にメソッドを使用することで、AST の構築を行う。コードの出力には、AST の構築が完了した VModule から、実装している endmodule メソッドを呼び出すことで出力することができる。

現時点では標準出力からのみの出力をサポートしており、コンパイルの完了と同時に標準出力より生成された Verilog のコードが出力される。生成構文は Verilog2001 に準拠し、論理合成可能な構文を主としたライブラリとして実装した。

† 岡山理科大学工学研究科システム科学専攻
‡ 岡山理科大学工学部情報工学科

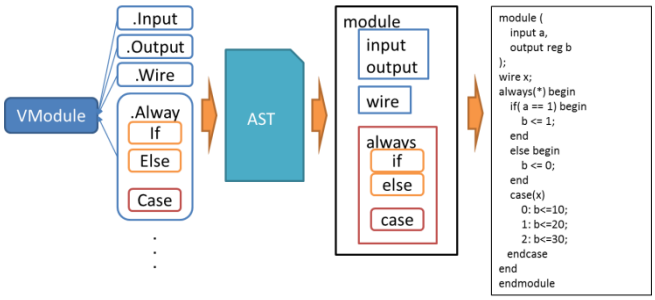

図 1 ハードウェア用 AST 構築の手順

表 1 提案システムの出力した性能

\Resource Process	FF	LUT	Memory LUT	BRAM
Synthesis	230(0.49%)	262(0.22%)	0(0.0%)	0(0.0%)
Implementation	230(0.49%)	262(0.22%)	0(0.0%)	0(0.0%)

4. 実験結果

実験として、提案システムで生成されたコードを FPGA 上に実装する。実装には Xilinx 製 FPGA の Zybo Z7-20 および、開発ソフト Vivado 2018.2 を用いた。表 1 に今回作成した山登り法の Rust から出力した Verilog のコードを実装した性能を示す。この回路では Synthesis(論理合成)および Implementation(配置配線)で使用される FPGA のリソースはどちらも FF(Flip-Flop)を 230 ブロック、LUT(LookUp Table)を 262 ブロック使用していた。Memory LUT や BRAM(Block RAM)はメモリを参照する回路ではなかったので使用されることは無かった。

5. おわりに

本稿では、Rust から Verilog へのトランスパイラを実装し、提案システムを用いて山登り法のアルゴリズムを Verilog のコードへ変換する実験を行った。その結果、変換された Verilog のコードは Synthesis、Implementation でき、Zybo Z7-20 内で使用されるリソースは FF が 230 ブロックで 0.49%、LUT が 262 ブロックで 0.22%であった。

今後は、様々なハードウェアを本ツールにより記述しコードの検証を進めていくとともに、機能拡張を行っていく必要がある。

参考文献

[1] N. D. Matsakis, F. S. Klock II, “The rust language”, ACM SIGAda Annual Conference on High Integrity Language Technology, Vol. 34, No.3, pp.103-104, 2014.
[2] “Verugent”, Available at: https://github.com/RuSys/Verugent/ Accessed: 22 August 2018.